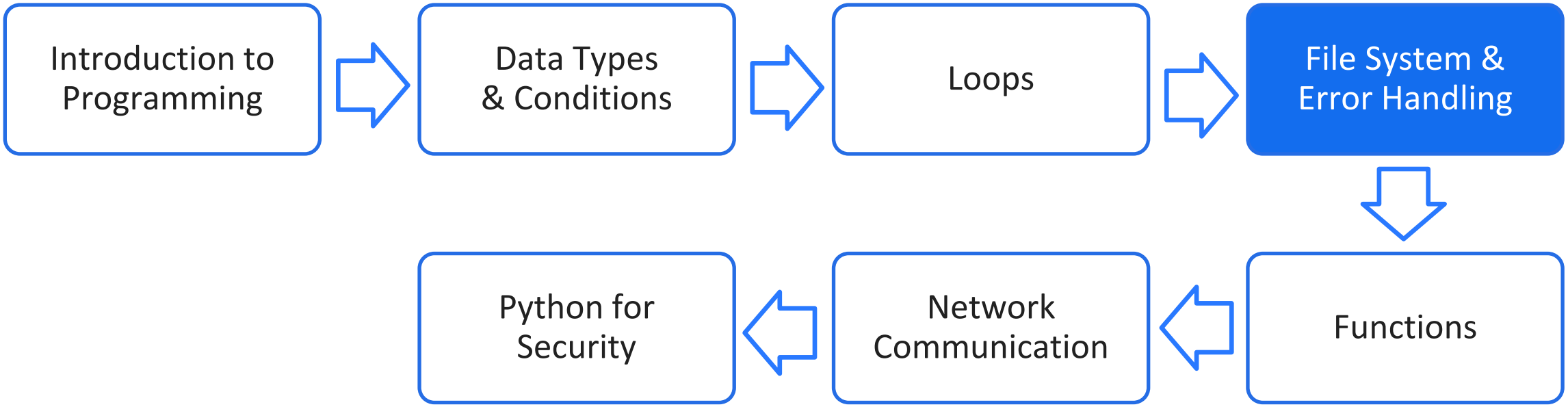
Cybersecurity Professional Program

File System & Error Handling

Introduction to Python for Security

Course Path


Introduction to Programming
Data Types & Conditions
Loops
File System & Error Handling
Functions
Network Communication
Python for Security

Objectives

This lesson covers how Python handles errors as well as how to open and write to files. The lesson also introduces the concept of Python modules and their uses.

- Error Handling
- File Manipulation
- Module Definition & Usage
- Log Parsing

File System & Error Handling

Error Handling

How Python Handles Errors

Python programs may encounter errors and raise exceptions.

If not handled, the program stops and presents an error.

Error handling can prevent a crash and allow the script to continue functioning.

```
num1 = 10
num2 = 0
div = num1 / num2
print(div)

num3 = 15
num4 = 20
sum = num3 + num4
print(sum)

==================================================================
<Output in here>

Traceback (most recent call last):
  File "C:/Users/PycharmProjects/error.py", line 3, in <module>
    div = num1/num2
ZeroDivisionError: division by zero

Process finished with the exit code 1
==================================================================
```

Try & Except

Error messages explain the reason for a crash.

try creates a code block to execute code that may create an error.

except handles the error in the ***try*** code.

```
try:
    num1 = 10
    num2 = 0
    div = num1 / num2
    print(div)

except:
    print("Cannot calculate it")
    num3 = 15
    num4 = 20
    sum = num3 + num4
    print(sum)

==========================================================
<Output in here>

Cannot calculate it
35

Process finished with the exit code 0
==========================================================
```

Common Exception Types

Exception Type	Explanation
Exception	Base class for all exceptions
StopIteration	Stops the ***next()*** iterator from continuing
SystemExit	Raised by the ***sys.exit()*** function
ArithmeticError	Occurs for numeric calculations
ZeroDivisionError	Division or modulo by zero
TypeError	Applies an operation to mismatched data types
KeyboardInterrupt	Generated by programmer/user

Error Handling Loops

Exception as error: Inserts an error message in a variable

Raise: Prints a custom error message

Else: Runs only if no exception occurred

Finally: Runs whether the code failed or succeeded

```
age = 13

try:
    if age < 18:
        raise Exception("The user is younger than 18.")
except Exception as error:
    print(f"Error: {error}")
else:
    print("No error has occurred.")
finally:
    print("This will always run.")

==================================================================
<Output in here>

Error: The user is younger than 18.
This will always run.

Process finished with the exit code 0
==================================================================
```

Error Handling

Lab PY-04-L1

Try & Except Practice

10–20 Min.

Mission

Use ***try*** and ***except*** to handle code errors.

Steps

- Request a number.
- Divide by zero.
- Handle the exception.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-04-L2

Error Handling

15–20 Min.

Mission

Practice handling errors that may occur in the code.

Steps

- Create an iteration of four loops.
- Request a number for multiplication.
- Ask the user for numbers.
- Handle potential errors.

Environment & Tools

- Python 3
- PyCharm

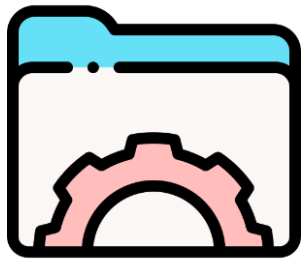
Related Files

- Lab document

File System & Error Handling

File Manipulation

Python & Files



- Python includes several built-in modules and functions for file handling.
- In Python, a file is categorized as either text (character sequence) or binary.
- Binary files are processed only by applications that work with the binary structure.

File Access Modes

Python requires a declaration of how a file is opened and with which permissions.

Mode	Description
r	Opens a file for reading only
b	Opens a file in binary format
a	Opens a file for appending; creates a new file if it does not exist
w	Opens a file for writing only and overwrites the existing file
+	Adds a read or write operation depending on the one selected

Opening Files

The ***open()*** function enables access to a file for the Python interpreter.

The ***open()*** function requires a file name and access mode.

The ***close()*** function must be added after ***open*** to be able to read the file and/or save changes made.

```
myfile = open("file.txt", "r")
x = myfile.read()
print(x)
myfile.close()
```

```
==========================================================
<Output in here>
this is line 1
this is line 2
this is line 3
this is line 4
this is line 5

Process finished with exit code 0
==========================================================
```

Writing to Files

The ***write()*** function writes content to an open file.

Content can be written only if the file was opened in the appropriate mode.

Read-only access raises an exception when ***write*** is used.

```
myfile = open("file.txt", "w")
myfile.write("When I write into the file; do I overwrite
the content?")
myfile.close()

myfile = open("file.txt", "r")
print(myfile.read())
myfile.close()

==================================================================
<Output in here>
When I write into the file; do I overwrite the content?

Process finished with exit code 0
==================================================================
```

Reading Files

The ***read()*** function allows reading the file according to the permission.

The ***readline()*** function is used to read a file line by line.

The ***readline([bytes])*** function can be used to read the number of bytes from the line.

```
myfile = open("file.txt", "r")
print(myfile.readline())
myfile.close()

myfile = open("file.txt", "r")
print(myfile.readline(5))
myfile.close()
```

```
==========================================================
<Output in here>
This is the first line of my file.

This

Process finished with exit code 0
==========================================================
```

Closing Files

After writing to a file, it is important to close it.

Not closing a file may cause a program to behave unexpectedly.

The ***close()*** function saves and closes a file.

```
myfile = open("file.txt", "w")
myfile.write("save & close")
myfile.close()

myfile = open("file.txt", "r")
print(myfile.read())
myfile.close()
```

```
==================================================================
<Output in here>
save & close

Process finished with exit code 0
==================================================================
```

Flush

The ***flush()*** function is used to save a file and continue editing it.

The file is saved in the system but remains open.

```
myfile = open("file.txt", "w")
myfile.write("save only")
myfile.flush()

myfile = open("file.txt", "r")
print(myfile.read())
myfile.close()


==================================================================
<Output in here>
save only


Process finished with exit code 0
==================================================================
```

With Open Statement

A ***with*** statement will create a context manager.

The ***open()*** function sets a new variable to handle the file.

The variable that contains the file is the one defined after ***as***.

```
with open("file.txt", "w") as file:
    file.write("with open statements, no closing needed")

with open("file.txt", "r") as file:
    data = file.read()
    print(data)


==================================================================
<Output in here>
with open statements, no closing needed


Process finished with exit code 0
==================================================================
```

File Manipulation

Lab PY-04-L3

Handling Files

10–20 Min.

Mission

Practice working with files and error handling.

Steps

- Create a file while using error handling.
- Create a ***try*** block.
- Create a loop to write text to the file.
- Break the loop if the word ***exit*** is input.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

File Manipulation

Lab PY-04-L4

Extracting Lines

10–20 Min.

Mission

Print text lines from a file.

Steps

- Receive a path and file name from the user.
- Open the specified file.
- Print each line.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Career Outcomes Reminder:
Resume & LinkedIn Review

1) Complete the Resume & LinkedIn Review module:
   Module subtopics:
   - ❑ Testing your career development knowledge

2) This module is located after “Welcome! Begin Here” in your cybersecurity course shell.

3) Connect with your SSM if you have questions.

File System & Error Handling

Module Definition & Usage

What Are Modules?

- Python source files
- Contain Python entities and functionalities
- Can be imported and used in Python files
- Used to break down large programs into smaller, more easily managed files

> Python has a large collection of importable modules that create the Python Standard Library.

Python *OS* Module

- The ***OS*** module in Python provides various ways of working with the operating system.
- It includes functions that enable Python to interface with programs running in different operating systems.

The ***OS*** module enables running system commands.

OS Functions

The ***os.system()*** function allows system operations and the execution of shell commands.

Output will be sent to the standard interpreter.

It is used by attackers in exploits or to bypass antivirus applications.

```
import os

os.system("ping 8.8.8.8")

==================================================================
<Output in here>

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51
Reply from 8.8.8.8: bytes=32 time=60ms TTL=51
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 6ms, Maximum = 8ms, Average = 7ms

Process finished with exit code 0
==================================================================
```

Methods

In comparison with functions, methods include two additional parameters.

A method is associated with an object or a class and can access data within a class.

```
import os

os.system("ping 8.8.8.8 >> pingTest.txt")

with open("pingTest.txt", "r") as file:
    for line in file:
        if "ms" in line:
            print("You have an internet connection.")
            break
        elif "unreachable" in line:
            print("You do not have an internet connection.")
        else:
            continue

==================================================================
<Output in here>
You have an internet connection

Process finished with exit code 0
==================================================================
```

Listing Directories

The ***os.listdir(path)*** function lists all the files in the specified directory.

The ***r*** before the directory instructs the interpreter to regard the characters that follow as a raw string.

```
import os

print(os.listdir())

==========================================
<Output in here>

['file', 'pingTest.txt', 'try.py', 'users’]

Process finished with exit code 0
==========================================
```

Advanced *OS* Module 1

The ***os.chdir()*** function changes the working directory.

The ***os.mkdir()*** function creates a directory.

The ***os.rmdir()*** function removes a directory.

The ***os.rename()*** function renames a file.

```
#change the current path
os.chdir(path)

#create a new directory
os.mkdir(newName)

#delete a directory
os.rmdir(deleteDirName)

#rename the file
os.rename(oldName.txt, newName.txt)
```

Advanced *OS* Module 2

The ***os.getcwd()*** function gets the location of the working directory.

The ***os.walk()*** function maps folders and files.

The ***os.stat()*** function provides information about files and directories and performs a statistical system call.

```
#get the path
os.getcwd()

#map all the files and directories
os.walk(folderPath)

#perform a stat system call on the given path
os.stat(folderPath)
```

Datetime

The ***datetime*** module includes classes for date and time.

It provides functions that handle the time, date, and time intervals.

Dates in ***datetime*** are objects, not strings.

```
import datetime

print("current date is{}".format(datetime.datetime.now()))
h = datetime.datetime.now().hour
m = datetime.datetime.now().minute
s = datetime.datetime.now().second
print("current date is {}:{}:{}".format(h,m,s))


==========================================================
<Output in here>

current date is 2022-05-03 19:37:07.282371
current date is 19:37:7

Process finished with exit code 0
==========================================================
```

Lab PY-04-L5

OS Module & Open Function

20–40 Min.

Mission

Save a ***ping*** output to a file with Python's ***OS*** module.

Steps

- Save command output to a file.
- Confirm network connectivity from the saved output.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-04-L6

OS Module

30–40 Min.

Mission

Create a directory and a file in that directory using the ***OS*** module.

Steps

- Create a ***while*** loop that requests a directory.
- Create the directory and navigate to it.
- Get a file name.
- Create the file.
- Write text and the date to the file.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Platform Module

The ***platform*** module is used to access data related to the OS and hardware.

Use the ***platform*** module to get information about a machine.

```
import platform

print("\n processer name is -> {}".format(platform.processor()))
print("\n python compiler is -> {}".format(platform.python_compiler()))
print("\n operation system is -> {}".format(platform.system()))
print("\n OS version is -> {}".format(platform.version()))
print("\n OS architecture is -> {}".format(platform.architecture()))
=================================================================
<Output in here>

processer name is -> Intel64 Family 6 Model 141 Stepping 1, Genuine Intel
python compiler is -> MSC v.1929 64 bit (AMD64)
operation system is -> Windows
OS version is -> 10.0.22000
OS architecture is -> ('64bit', 'WindowsPE’)

Process finished with exit code 0
=================================================================
```

Random Module

The ***random*** module enables the generation of pseudo-random numbers.

The module includes various methods.

random.randint(x,y) returns a random integer in the range that appears in parentheses.

```
import random

While True:
    user_guess = int(input("Guess a number between 1 and 10: "))
    randG = random.randint(1,10)
    if randG != user_guess:
        print("Wrong guess, the number is: {}".format(randG))
    else:
        print("Congrats, you guessed the number!")
==================================================================
<Output in here>

Guess a number between 1 and 10: 5
Wrong guess, the number is: 9
Guess a number between 1 and 10: 4
Congrats, you guessed the number!

Process finished with exit code 0
==================================================================
```

File System & Error Handling

Log Parsing

Log File Parsing 1

Use ***open()*** to open a file.

Use a ***for*** loop to read the file's text line by line.

Use ***split()*** to choose which information to remove.

```
str1 = "hello world"

newStr = str1.split(" ")
print("List result from using split: {}".format(newStr))
for word in newStr:
    if word == "world":
        print("Found the word 'world'!")
        break
==================================================
<Output in here>

List result from using split: ['hello', 'world']
Found the word 'world'!

Process finished with exit code 0
==================================================
```

Log File Parsing 2

Each string in the document is manipulated and split.

Parts of the string are selected to construct parsed logs.

The selected parts are concatenated to the desired structure.

```
with open("log.txt","r") as file:
  for line in file:
     if "ftp" in line:
       parsed = line.split(" ")
       print("Date:{}, OS:{}".format(parsed[0],parsed[1]))

==========================================================
<Output in here>

Date: 01/01/22, OS: macOS
Date: 01/01/22, OS: win10

Process finished with exit code 0
==========================================================
```

Log Parsing

Lab PY-04-L7

Encoding & Decoding the Secret

15–20 Min.

Mission

Encode and decode a Base64 secret using Python, and print the original message to the console.

Steps

- Import the Base64 library.
- Get a Base64 secret from the user.
- Encode the secret.
- Print the encoded secret.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-04-L8

Copying Files

10–20 Min.

Mission

Practice working with the ***OS*** module.

Steps

- Receive parameters.
- Run the ***copy*** command.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Log Parsing

Lab PY-04-L9

Tasks & Questions

15–20 Min.

Mission

Answer questions on the subject taught in class.

Steps

- Answer the questions in the lab document.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

TDX Arena Lab Homework

Mission

Use TDX Arena for additional practice opening and analyzing files in Python.

Steps

Sign in to the **TDX Arena** platform.

Navigate to the **Practice Arena**.

Navigate to the **Python Programming** course.

Select ***PY06 Data Analysis***.

Select the ***Opening & Analyzing Files*** lab.

Complete the homework **before** the next class.

Complete any labs or challenges you **did not finish** in class.

Opening & Analyzing Files

TDX Arena Lab Homework

Mission

Use TDX Arena to better understand regular expressions in the context of Python.

Steps

Sign in to the **TDX Arena** platform.

Navigate to the **Practice Arena**.

Navigate to the **Python Programming** course.

Select ***PY06 Data Analysis***.

Select the ***Regular Expressions*** lab.

Complete the homework **before** the next class.

Complete any labs or challenges you **did not finish** in class.

Regular Expressions

Thank You

Questions?